An Isabelle formalization of protocol-independent secrecy with an application to e-commerce

Frédéric Blanqui

An Isabelle formalization of protocol-independent secrecy with an application to e-commerce

Frédéric Blanqui

Laboratoire d'Informatique de l'École Polytechnique
91128 Palaiseau Cedex, France
http://www.lix.polytechnique.fr/~blanqui/

Abstract. A protocol-independent secrecy theorem is established and applied to several non-trivial protocols. In particular, it is applied to protocols proposed for protecting the computation results of free-roaming mobile agents doing comparison shopping. All the results presented here have been formally proved in Isabelle by building on Larry Paulson's inductive approach. This therefore provides a library of general theorems that can be applied to other protocols.

1 Introduction

Cryptographic protocols are intended to ensure properties like secrecy, authentication, anonymity, integrity or non-repudiability. Initially proposed for securing communications, they have been more recently proposed for protecting the computation results of free-roaming mobile agents doing comparison shopping [26]. Experience shows that even simple protocols are difficult to set correctly [14]. Their correctness clearly needs to be checked by mechanical tools. But this cannot be an easy task. For instance, secrecy has been shown undecidable even under very weak assumptions [8]. And correctness is always with respect to a given model. A protocol correct in a model may be incorrect in a richer model [23]. As for the effectiveness of cryptography, which is often assumed perfect in the formal versions of protocols, fortunately, it may be precisely related to the one of real protocols [11]. Although we cannot expect formal methods to be able to deal with every possible problem (like [25] for instance), yet they provide better confidence and may serve in finding protocol-specific flows.

Many different approaches have been proposed so far. Approaches based on model-checking are quite effective but the state explosion forces the model to be kept simple, typically by limiting the number of agents and other parameters to one or two although, in some cases, checking can be done in a finite model [13]. Approaches based on proof assistants allow a finer and more general modelization but require much effort, typically of several days or weeks, although the use of automated theorem provers may be very effective too [6]. Proof-assistants (Isabelle [18], PVS [24], etc.) can establish protocol-independent theorems whose conditions could be proved by means of specialized automatic

tools, providing a really complete certification. In this spirit, Millen and Rueß formalized a protocol-independent secrecy theorem [15] in PVS [16] and later developed with Cortier an automatic proof search procedure [7].

In this paper, we develop in Isabelle a more general secrecy theorem based on Paulson's inductive approach [21], which has several advantages over the previous approach. First, we require no additional information (*spell* or *state events*) in order to be more general and also compatible with the libraries already implemented in Isabelle by Paulson and Bella. Second, we establish a first secrecy theorem (Section 3) which only involves the Isabelle theory of messages, and thus is independent of any protocol formalization. It is based on the simple and intuitive notion of *guarded messages* which, as opposed to the notion of *coideal* used in [15], distinguishes between what must be kept secret (a key or a nonce `n`) from how this secret is ensured (the set `Ks` of the keys whose inverses are used for encrypting `n`). It simply says that, if someone can get `n` by decomposing and decrypting a set of guarded messages, then he must also be able to get one of the key of `Ks`. Therefore, if the keys of `Ks` are not compromised, then the spy cannot get `n`.

Then, we establish a second theorem (Section 4) showing that, for proving the guardedness of a nonce or a key `n` used in a protocol, and hence that `n` is kept secret, it is sufficient to prove that each rule of the protocol indeed preserve the guardedness of `n`, without having to care about what the spy can do (which is taken into account once and for all in the proof of the theorem). This departs from proofs in strand spaces [10] or with coideals [7] where it is necessary to step back into the protocol rules, including the rules for the spy, in order to explore every possibility of how certain message fields could have been published.

We applied our results to several well-known protocols: Needham-Schoeder-Lowe [17,14], Otway-Rees [19] and Yahalom [1]. We also verified two protocols proposed by Asokan, Gülcü and Karjoth [2] for protecting the computation results of free-roaming mobile agents doing comparison shopping. These have never been formally certified before. The properties claimed by the authors (see Section 5) not only include confidentiality properties but also non-repudiability and integrity properties. This shows a new application of Isabelle.

The paper is organized as follows. In Section 2, we quickly present Paulson's inductive approach [21]. In Section 3, we present our notion of *guarded message* for proving secrecy. In Section 4, we introduce a more precise formalization of protocols and show how it helps to prove guardedness. In Section 5, we present the protocols P1 and P2 proposed in [2] for protecting the computation results of free-roaming agents. In Section 6, we explain the formal proof of their correctness.

All the Isabelle files are freely available on our web page.

2 Paulson's inductive approach

The inductive approach has been introduced by Paulson [21] and applied to many non-trivial protocols within the generic proof assistant Isabelle [18]: Kerberos IV

[3], TLS [22], SET [20], etc. All these results are part of the Isabelle distribution and can be freely used to certify other protocols.

In this approach, a protocol is represented as the set of all the possible sequences of events that can occur by following the protocol steps and by having a spy able to send fake messages built from the analysis of past traffic. An infinite number of agents is assumed:

```
datatype agent = Server | Friend nat | Spy
```

Messages are represented as the elements of the following inductive data type:

```
datatype msg = Number nat (* guessable *) | Nonce nat (* not guessable *)
| Agent agent | Key nat | Hash msg | {|msg, msg|} | Crypt key msg
```

This has several important consequences:

- Encryption is assumed to be perfect: decryption can only be done by using the inverse of the key used for encryption. Within a public-key infrastructure, the inverse of the public key of an agent `A`, `pubK A`, is its private key `priK A`. Otherwise, each agent `A` is assumed to have a symmetric key `shrK A` whose inverse is itself.
- Hashing is collision-free: two distinct messages give two distinct hash codes. This comes from the fact that the constructors of an inductive data type are injective.
- Each kind of message is recognizable and hence cannot be confused with another one: an agent name `Agent A` is distinct from a key `Key K` or an encrypted message `Crypt K X`, etc. This comes from the fact that the constructors of an inductive data type have distinct images. This implies that encryption schemes for which there are relations between encrypted messages like XOR (`Crypt K (Crypt K X) = X`) or RSA (`Crypt K (Crypt K' X) = Crypt K' (Crypt K X)`) cannot be formalized.

Then, Paulson introduces three important functions:

- The function `parts H` returns the set of all the actual sub-components of a set `H` of messages (`X` is not a sub-component of `Hash X`):

```
consts parts :: "msg set ⇒ msg set"
inductive "parts H" intros
Inj: "X ∈ H ⟹ X ∈ parts H"
Fst: "{|X,Y|} ∈ parts H ⟹ X ∈ parts H"
Snd: "{|X,Y|} ∈ parts H ⟹ Y ∈ parts H"
Body: "Crypt K X ∈ parts H ⟹ X ∈ parts H"
```

- The function `analz H` returns the set of all the sub-components that can be obtained by decomposing and decrypting (if the necessary key has itself been obtained) all the messages of the set `H`:

```
consts analz :: "msg set => msg set"
inductive "analz H" intros
Inj: "X ∈ H ⟹ X ∈ analz H"
Fst: "{|X,Y|} ∈ analz H ⟹ X ∈ analz H"
Snd: "{|X,Y|} ∈ analz H ⟹ Y ∈ analz H"
```

```
Decrypt: "[[Crypt K X ∈ analz H; Key(invKey K) ∈ analz H]]
  ⟹ X ∈ analz H"
```

- The function `synth` returns the set of all the messages that can be synthesized from a set `H` of messages (new nonces and new keys cannot be synthesized):

```
consts synth :: "msg set ⇒ msg set"
inductive "synth H" intros
Inj: "X ∈ H ⟹ X ∈ synth H"
Agent: "Agent agt ∈ synth H"
Number: "Number n ∈ synth H"
Hash: "X ∈ synth H ⟹ Hash X ∈ synth H"
Pair: "[[X ∈ synth H; Y ∈ synth H]] ⟹ {|X,Y|} ∈ synth H"
Crypt: "[[X ∈ synth H; Key K ∈ H]] ⟹ Crypt K X ∈ synth H"
```

Finally, the fact that an agent `A` sends to another agent `B` a message `X` is represented by the event `Says A B X`.

As an example, we give the formalization of the Needham-Schroeder-Lowe protocol [14]:

1. $A \rightarrow B : \{Na, A\}_{Kb}$
2. $B \rightarrow A : \{Na, Nb, B\}_{Ka}$
3. $A \rightarrow B : \{Nb\}_{Kb}$

```
consts nsl :: "event list set"
inductive nsl intros
Nil: "[] ∈ nsl"
Fake: "[[evs ∈ nsl; X ∈ synth(analz(spies evs))]]
  ⟹ Says Spy B X # evs ∈ nsl"
NS1: "[[evs ∈ nsl; Nonce NA ∉ used evs]]
  ⟹ Says A B (Crypt (pubK B) {|Nonce NA, Agent A|}) # evs ∈ nsl"
NS2: "[[evs ∈ nsl; Nonce NB ∉ used evs;
  Says A' B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs]]
  ⟹ Says B A (Crypt (pubK A) {|Nonce NA,Nonce NB,Agent B|}) # evs ∈ nsl"
NS3: "[[evs ∈ nsl; Says A B (Crypt (pubK B) {|Nonce NA,Agent A|}) ∈ set evs;
  Says B' A (Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}) ∈ set evs]]
  ⟹ Says A B (Crypt (pubK B) (Nonce NB)) # evs ∈ nsl"
```

The first two rules are in any protocol. `Nil` means that the empty trace `[]` is in the protocol. `Fake` means that the spy can send fake messages built from the analysis of past traffic. The next rules are the rules of the protocol. `set evs` is the set of all the messages sent so far. `used evs` denotes the parts of all the messages sent so far. `ev # evs` is the list of head `ev` and tail `evs`.

Then, one may prove for example that, if `B` sends the message of step 2 and receives the message of step 3, and if the private keys of `A` and `B` are not compromised (`A` and `B` do not belong to the set `bad` of bad agents), then the protocol must have been initiated by `A`:

```
lemma "[[A ∉ bad; B ∉ bad; evs ∈ nsl;
Says B A (Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}) ∈ set evs;
```

```
Says A' B (Crypt(pubK B) (Nonce NB)) ∈ set evs⟧
⟹ Says A B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs"
```

As most authentication properties, it relies on the fact that the nonces used by the agents and encrypted with their public keys are not known by the spy:

```
lemma "⟦A ∉ bad; B ∉ bad; evs ∈ nsl;
Says A B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs⟧
⟹ Nonce NA ∉ analz(spies evs)"
```

This is why, in the following, we concentrate on secrecy proofs.

3 Guarded messages

The idea of the protocol-independent secrecy theorem is simple and intuitive. In any protocol, secrecy is ensured by using cryptography and by making sure that the key used for the encryption is indeed safe. We simply turn this into a formal definition, the notion of *guarded message*, and formally prove that getting the secret indeed requires to know the key used for encryption (normally safe).

Definition 1 (Guarded messages). *A nonce or a key* `n` *is* guarded *in a message* `X` *by a set of keys* `Ks` *if every occurrence of* `n` *in* `X` *is inside a sub-message encrypted by the inverse of a key of* `Ks`*. We denote by* `guard n Ks` *the set of messages in which* `n` *is guarded by* `Ks`*.*

```
consts guard :: "nat ⇒ key set ⇒ msg set"
inductive "guard n Ks" intros
No_Nonce: "Nonce n ∉ parts X ⟹ X ∈ guard n Ks"
Guard: "invKey K ∈ Ks ⟹ Crypt K X ∈ guard n Ks"
Crypt: "X ∈ guard n Ks ⟹ Crypt K X ∈ guard n Ks"
Pair: "⟦X ∈ guard n Ks; Y ∈ guard n Ks⟧ ⟹ {|X,Y|} ∈ guard n Ks"
```

The secrecy theorem can then be stated as follows:

Theorem 2 (Secrecy). *Let* `G` *be a set of messages where* `Nonce n` *is guarded by a set of keys* `Ks`*. If* `Nonce n` *belongs to* `analz G` *then there exists a key* `K` *in* `Ks` *that belongs to* `analz G`*:*

```
theorem Guard_invKey: "⟦Nonce n ∈ analz G; Guard n Ks G⟧
⟹ ∃ K. K ∈ Ks & Key K ∈ analz G"
```

Proof. Although this result may seem obvious, it is not straightforward to prove in Isabelle or in any other proof assistant. We explain how our formal proof proceeds.

First, only a finite part of `G` needs to be analyzed to get `Nonce n`. Therefore, we can assume that `G` is finite. We then associate to any finite set `G` a measure μ(`G`) which is the number of encrypted messages in `G`. We can then reason by induction on μ(`G`).

Second, to help reason about `analz`, we proved the following very useful decomposition theorem: `analz G` = `pparts G` ∪ `analz` (`kparts G`), where `pparts`

G is all the pairs of G, their components that are themselves pairs and so on, and kparts G is all the components that are not pairs.

Now, since all the messages in G are guarded, to get Nonce n, there must be an encrypted message Crypt K Y in kparts G such that Key (invKey K) belongs to kparts G also: we must decrypt at least one encrypted message for reaching Nonce n. If K is a key of Ks then we are done. Otherwise, let H = kparts G \ {Crypt K Y}. Then, by definition of analz, Nonce n must belong to analz (H ∪ {Y}) whose measure is strictly smaller than the one of G. Therefore, we can conclude by induction hypothesis. □

Our notion of guardedness has several advantages over the notion of coideal of Millen and Rueß [15]. First, it is defined inductively while a coideal is defined as the complement of an ideal [9] (which is defined inductively). Second, it is more general in the sense that the coideal of {Nonce n} ∪ Ks is included in guard n Ks but not the converse. For instance, while Key K is guarded by {K} in Crypt (invKey K) (Key K), it does not belong to the coideal of {K}. This is due to the fact that there is no separation between what must be kept secret (n in guard n Ks) and how it must be kept secret (Ks in guard n Ks). Third, guardedness enjoys a nice monotonicity property:

```
lemma guard_extend: "Ks ⊆ Ks' ⟹ guard n Ks ⊆ guard n Ks'"
```

which is very useful in the case of protocols like Yahalom where a nonce must be guarded by a session key issued by a server (see below). This is not the case with coideals. For instance, {|Crypt (pubK A) (Nonce n), Key K|}, in which Nonce n is guarded by {Nonce n, priK A}, hence by {Nonce n, priK A, Key K} too, belongs to the coideal of {Nonce n, priK A} but not to the coideal of {Nonce n, priK A, Key K}. So, guardedness is a finer and more intuitive notion than the one of coideals.

In the case of the Needham-Schroeder-Lowe protocol, one can formally prove that NA and NB are both guarded by {priK A,priK B}. Therefore, after our theorem, if the private keys of A and B are not compromised then the spy cannot have access to NA and NB.

```
lemma Guard_NA: "⟦evs ∈ nsl; A ∉ bad; B ∉ bad;
  Says A B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs⟧
  ⟹ Guard NA {priK A,priK B} (spies evs)"
lemma Guard_NB: "⟦evs ∈ nsl; A ∉ bad; B ∉ bad;
  Says B A (Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}) ∈ set evs⟧
  ⟹ Guard NB {priK A,priK B} (spies evs)"
```

We applied our theorem to other well known protocols: the Otway-Rees symmetric-key protocol [19] and the Yahalom symmetric-key protocol [1]. By doing so, we noticed a much shorter development time compared with the direct proofs done by Paulson [21]. Of course, this is not as fast as automatic tools like TAPS [6] but we expect to define tactics to automate these guardedness proofs.

The Yahalom protocol is interesting since it is a simple case of dependency between secrets [23]. We recall the informal definition of the protocol (S denotes the server):

1. $A \rightarrow B : A, Na$
2. $B \rightarrow S : B, \{A, Na, Nb\}_{Kb}$
3. $S \rightarrow A : \{B, K, Na, Nb\}_{Ka}, \{A, K\}_{Kb}$
4. $B \rightarrow A : \{A, K\}_{Kb}, \{Nb\}_{K}$

One can prove that the session key `K` is guarded by the keys of `A` and `B`, and that the nonce `NB` is guarded by the keys of `A` and `B` and all the session keys issued by the server:

```
lemma Guard_KAB: "⟦evs ∈ ya; A ∉ bad; B ∉ bad;
  Says Server A {|Crypt (shrK A) {|Agent B, Key K, Nonce NA, Nonce NB|},
  Crypt (shrK B) {|Agent A, Key K|}|} ∈ set evs⟧
  ⟹ Guard K {shrK A,shrK B} (spies evs)"
lemma Guard_NB: "⟦evs ∈ ya; A ∉ bad; B ∉ bad; Says B Server
  {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|} ∈ set evs⟧
  ⟹ Guard NB ({shrK A,shrK B} ∪ keys A B NA NB evs) (spies evs)"
keys A B NA NB evs = {K. Says Server A {|Crypt (shrK A) {|Agent B, Key K,
  Nonce NA, Nonce NB|}, Crypt (shrK B) {|Agent A, Key K|}|} ∈ set evs}
```

In our formalization, the server may issue several session keys for the same request by repeating step 3. This may be avoided by checking whether step 3 has already been done. Note also that the set of keys `Ks` used here takes care of two different states of the protocol: if no session key has been issued yet then, indeed, only the keys of `A` and `B` are sufficient for protecting `NB`.

4 Guardedness proofs

In order to ease guardedness proofs, we have developed other theorems inspired by the work of Millen and Rueß [15]. The idea is to reduce the proof for all possible traces to a proof that all the protocol rules preserve the guardedness condition. This requires to introduce a more precise formalization of the notion of protocol.

Definition 3 (Protocol rule). *A* message pattern *is a message with variables of distinct types for agents, nonces, keys, etc.* Event patterns *are defined similarly. A* substitution `s` *is an application which associates an agent to each agent variable, a nonce to each nonce variable, etc. Replacing each variable of a message pattern* `X` *by its image in a substitution* `s` *gives a message denoted by* `apm s X`*. For an event pattern* `ev`*, the substitution is denoted by* `ap s ev`*.*

A rule `R` *is a pair made of a set of event patterns* `fst R`*, the* preconditions*, and an event pattern* `snd R`*. The set of* new nonces *of a rule* `R`*,* `newn R`*, is the set of nonce variables that occur in* `snd R` *and in no precondition.*

Now, a protocol will be seen as a set of rules:

```
types proto = "(event set * event) set"
```

Then, the traces generated by a protocol are inductively defined as follows:

```
consts tr :: "proto ⇒ event list set"
inductive "tr p" intros
```

```
Nil: "[] ∈ tr p"
Fake: "⟦evs ∈ tr p; X ∈ synth(analz(spies evs))⟧
  ⟹ Says Spy B X # evs ∈ tr p"
Rule: "⟦evs ∈ tr p; R ∈ p; ok evs R s⟧ ⟹ ap s (snd R) # evs ∈ tr p"
```

An s-instance of a rule R can be added to a trace evs if the s-instance of every precondition occurs in evs and the s-instance of every new nonce has not been used yet:

```
ok evs R s = (∀ x. x ∈ fst R ⟶ ap s x ∈ set evs)
  & (∀ n. n ∈ newn R ⟶ ap s n ∉ used evs)
```

Definition 4 (Freshness). *We will denote by* `fresh R' s' n Ks evs` *the fact that a nonce* `n` *is introduced in a trace* `evs` *by an* `s'`*-instance of a rule* `R'` *in which it is guarded by a set of keys* `Ks`*.*

Theorem 5 (Guardedness). *Assume that all the rules of the protocol preserve the guardedness of* `n` *w.r.t.* `Ks` *if the keys of* `Ks` *are safe:*

```
preserv p n Ks = ∀ evs R' s' R s. evs ∈ tr p ⟶ fresh R' s' n Ks evs
  ⟶ Guard n Ks (spies evs) ⟶ safe Ks (spies evs)
  ⟶ ok evs R s ⟶ ap s (snd R) ∈ guard n Ks
safe Ks (spies evs) = ∀ K. K ∈ Ks ⟶ K ∉ analz(spies evs)
```

Then, if `n` *is also guarded in the initial knowledge of the spy, we get that* `n` *is guarded in every possible trace:*

```
theorem preserv_Guard: "⟦preserv p n Ks; evs ∈ tr p; fresh R' s' n Ks evs;
  safe Ks (spies evs); Guard n Ks (init Spy)⟧ ⟹ Guard n Ks (spies evs)"
```

4.1 Examples

As an example, let us see the case of the Otway-Rees protocol [19]:

1. $A \rightarrow B : \{Na, A, B, \{Na, A, B\}_{Ka}\}$
2. $B \rightarrow S : \{Na, A, B, \{Na, A, B\}_{Ka}, \{Na, Nb, A, B\}_{Kb}\}$
3. $S \rightarrow B : \{Na, \{Na, K\}_{Ka}, \{Nb, K\}_{Kb}\}$
4. $B \rightarrow A : \{Na, \{Na, K\}_{Ka}\}$

The formal definition of the rules are:

```
OR1 = ({},
  Says A B {|Nonce NA, Agent A, Agent B,
  Crypt (shrK A) {|Nonce NA, Agent A, Agent B|}|})
OR2 = ({Says A' B {|Nonce NA, Agent A, Agent B, X|}},
  Says B Server {|Nonce NA, Agent A, Agent B, X,
  Crypt (shrK B) {|Nonce NA, Nonce NB, Agent A, Agent B|}|})
OR3 = ({Says B' Server {|Nonce NA, Agent A, Agent B,
  Crypt (shrK A) {|Nonce NA,Agent A, Agent B|},
  Crypt (shrK B) {|Nonce NA, Nonce NB, Agent A, Agent B|}|}},
  Says Server B {|Nonce NA, Crypt (shrK A) {|Nonce NA, Key KAB|},
  Crypt (shrK B) {|Nonce NB, Key KAB|}|})
```

```
OR4 = ({Says B Server {|Nonce NA, Agent A, Agent B, X,
  Crypt (shrK B) {|Nonce NA, Nonce NB, Agent A, Agent B|}|},
  Says S B {|Nonce NA, Y, Crypt (shrK B) {|Nonce NB, Key KAB|}|}},
  Says B A {|Nonce NA, Y|})
```

Let us prove the preservation property. We only detail the case of NB, the case of NA is similar. Assume that NB has been introduced by an instance of OR2 and is guarded by {shrK B}.

OR1: Assuming that NA' is a new nonce, we must prove that NB is guarded in P = {|Nonce NA', Agent A', Agent B', Crypt (shrK A') {|Nonce NA', Agent A', Agent B'|}|}. Since NA' is new, it cannot be equal to NB. Guardedness is therefore immediate since NB does not occur in P.

OR2: Assuming that NB' is a new nonce and that NB is guarded in {|Nonce NA', Agent A', Agent B', X|}, we must prove that NB is guarded in {|Nonce NA', Agent A', Agent B', X, Crypt (shrK B') {|Nonce NA', Nonce NB', Agent A', Agent B'|}|}. Again, since NB' is new, it cannot be equal to NB. Therefore, we are left with the case when NA' is equal to NB. But, since NB is guarded in {|Nonce NA', Agent A', Agent B', X|}, NA' cannot be equal to NB and we are done.

OR3: Assuming that NB is guarded in P = {|Nonce NA', Agent A', Agent B', Crypt (shrK A') {|Nonce NA',Agent A',Agent B'|}, Crypt (shrK B') {|Nonce NA',Nonce NB',Agent A',Agent B'|}|}, we must prove that NB is guarded in Q = {|Nonce NA', Crypt (shrK A') {|Nonce NA',Key KAB|}, Crypt (shrK B') {|Nonce NB', Key KAB|}|}. Since NB is guarded in P, NA' cannot be equal to NB because NA' is not guarded in P. Therefore, NB is guarded in Q.

OR4: Assuming that NB is guarded in {|Nonce NA',Agent A',Agent B',X, Crypt (shrK B') {|Nonce NA', Nonce NB', Agent A', Agent B'|}|} and in P = {|Nonce NA',Y,Crypt (shrK B') {|Nonce NB', Key KAB|}|}, we must prove that NB is guarded in {|Nonce NA',Y|}. Since NB is guarded in P, it cannot be equal to NA' and it is guarded in Y also. Therefore, NB is guarded in Q.

With this protocol, the preservation of guardedness does not require any extra property but, in general, it is necessary to have unicity lemmas [21,6]. Let us see for instance the Needham-Schroeder-Lowe protocol. Assume that NA has been introduced by an instance of NS1 and is guarded by {priK A, priK B}. The case of NB is similar.

NS1: Assuming that NA' is a new nonce, we must prove that NA is guarded in P = Crypt (pubK B') {|Nonce NA', Agent A'|}. Since NA' is new, it cannot be equal to NA. Thus, NA is guarded in P.

NS2: Assuming that NB' is a new nonce and that NA is guarded in Crypt (pubK B') {|Nonce NA', Agent A'|}, we must prove that NA is guarded in P = Crypt (pubK A') {|Nonce NA', Nonce NB', Agent B'|}. Again, since NB' is new, it cannot be equal to NA. So, we are left with the case when NA' = NA. But then, the trace contains two messages: Crypt (pubK B') {|Nonce NA, Agent A'|} and Crypt (pubK B) {|Nonce NA, Agent A|} for NA has been introduced with NS1.

Since NA is not known to the spy (the trace is assumed to be guarded), this cannot happen: we must have A = A' (see below). Hence, NA is guarded in P.

NS3: Assuming that NA is guarded in Crypt (pubK B') {|Nonce NA', Agent A'|} and in Crypt (pubK A') {|Nonce NA', Nonce NB', Agent B'|}, we must prove that it is guarded in P = Crypt (pubK B') (Nonce NB'). Assume that NB' = NA. Then, the trace contains two messages: Crypt (pubK A') {|Nonce NA', Nonce NA, Agent B'|} and Crypt (pubK B) {|Nonce NA, Agent A|} for NA has been introduced with NS1. Since NA is not known to the spy, this cannot happen (see below). Therefore, we must have NB' ≠ NA and NA is guarded in P.

To prove the guardedness, we used the following two unicity lemmas, which are easily proved:

```
lemma "⟦evs ∈ nsl; Crypt (pubK B) {|Nonce NA, Agent A|} ∈ parts(spies evs);
  Crypt (pubK B') {|Nonce NA, Agent A'|} ∈ parts(spies evs);
  Nonce NA ∉ analz (spies evs)⟧ ⟹ A=A' & B=B'"
lemma "⟦evs ∈ nsl; Crypt (pubK B) {|Nonce NA, Agent A|} ∈ parts(spies evs);
  Crypt(pubK B') {|Nonce NA', Nonce NA, Agent A'|} ∈ parts(spies evs)⟧
  ⟹ Nonce NA ∈ analz (spies evs)"
```

Similar lemmas hold for NB. They can be seen as two distinct instances of the same unicity lemma: if two rules that should introduce new nonces in fact introduce the same nonce in two guarded messages, and if this nonce is not known to the spy, then the two messages must be equal.

4.2 Comparaison with Cortier, Millen and Rueß' work

In [15], the proof of unicity lemmas is avoided by introducing additional information in the definition of protocols, the *spell events*, that indicate the nonces and keys that must be kept secret and the agents allowed to have access to them, and by enforcing the disjointness of spell events.

The guardedness proofs often follow the same pattern and use the same lemmas (that a new nonce is distinct from the nonce for which we want to prove the guardedness condition, the unicity lemmas, etc.). We therefore expect to turn this into an Isabelle tactic which would try to prove the guardedness automatically by using these lemmas.

Cortier, Millen and Rueß [7] define a search procedure for proving a similar property, the *occultness*. But, since it does not take into account the origin of a secret (the fresh predicate in our formalization), it needs to step back into the protocol rules and the Fake rule of the spy to get sufficient information for concluding, and this may not terminate. Furthermore, they require yet other information in the definitions of protocols, the *state events*, whose usefulness for occultness proofs is not clear.

5 Protocols P1 and P2

As an important and new application, we used our theorem to formally certify in Isabelle the correctness of some of the protocols proposed by Asokan, Gülcü

and Karjoth in [2] for protecting the computation results of free-roaming agents. These protocols tolerate collusion between servers and unfixed itineraries. They formalize and extend protocols proposed by Yee [26]. Application areas of these protocols include comparison shopping, bidding and network routing [5]. Following the authors, we will use the vocabulary of comparison shopping, hence using the word "shop" to denote a server, and the word "offer" to denote the answer of a shop to an agent's request.

These protocols are not concerned with the security problems raised by the use of mobile agents: aside from the correctness of the implementations of servers and mobile agents, that servers indeed execute the mobile agents code, and that servers and agents are indeed protected against malicious agents [26,4,12].

The jump of an agent from a server S_1 to a server S_2 can be seen as S_1 sending to S_2 a message representing the state of the agent. It therefore fits with our model. The difficulty here is that the itinerary is not known *a priori*: the set of servers to be visited can be chosen and extended by the agent itself. This is the case when an agent is programmed to go to some information servers which may provide it with addresses of shops or other information servers.

Asokan, Gülcü and Karjoth propose four protocols called P1, P2, P3 and P4 depending on the properties one would like and the available infrastructure. P1 and P2 require a public-key infrastructure. In P1, the author of an offer is not kept secret and the integrity of data is publicly verifiable while, in P2, the author of an offer can only be known to the owner of the agent. P3 and P4 do not assume a public-key infrastructure and use message authentication codes (MAC) instead. These last two protocols do not ensure non-repudiability. We formally proved all the properties claimed by the authors for P1 and P2. We left for future work the proof of P3 and P4.

All the protocols have a common structure: first, the owner of the agent sends his agent to the first server and, second, each server receiving the agent sends it to the next server after having added his own offer, this last step being repeated until the agent comes back to its owner. The difference between the protocols lie in the way the messages containing the offers are built.

For describing the state of the agent in P1 and P2, we adopted the message format {|`Agent A, Number r, I, L`|} where:

- `Agent A` is the owner of the agent,
- `Number r` is the agent's request,
- `I` is the list of servers to be visited,
- `L` is the list of offers collected so far.

We choose to send the request and the list of servers to be visited in the clear. This is not specified in [2] but we may assume that, in practice, the request is implemented as a public data and the itinerary as a private data. However, a malicious server executing the agent could easily know about the itinerary, even though some encryption is used. So, there is no point in hiding this information.

On the other hand, we must keep in mind that a malicious server can get important information from the itinerary stored in the agent and the way the offers are stored also. Indeed, assume that two servers S_1 and S_2 collude. If S_1

sends to S_2 the itinerary and the offers of the agent when the agent was at S_1, then S_2 can try to guess to whom belong the offers. As suggested in [2], a way to make this more difficult is to shuffle the offers after each new offer.

Furthermore, even if a server does not try to alter the offers collected so far by an agent, it may not give its best offer. Indeed, if it succeeds to know the offers collected by the agent so far, it may give an offer which is just a little bit better, but not as good as it could. This looks like a Vickery auction (the highest bidder pays the second highest bid) but upside-down: the best server offers the second lowest price [26].

Then, we formalized P1 and P2 as follows:

```
consts p :: "event list set"
inductive p intros
Nil: "[] ∈ p"
Fake: "⟦evs ∈ p; X ∈ synth(analz(spies evs))⟧
  ⟹ Says Spy B X # evs ∈ p"
Req: "⟦evs ∈ p; Nonce n ∉ used evs; I ∈ agl⟧
  ⟹ Says A B (reqm A r n I B) # evs ∈ p"
Prop: "⟦evs ∈ p; I ∈ agl; J ∈ agl; Says A' B {|Agent A,Number r,I,L|}
  ∈ set evs; isin (Agent C, app (J, del (Agent B, I))); Nonce ofr
  ∉ used evs⟧ ⟹ Says B C (prom B ofr A r P I L J C) # evs ∈ p"
```

`Nil` and `Fake` are the usual rules for the empty trace and the spy respectively. `Req` corresponds to the first step, the sending of the agent by its owner. We use an unguessable message, `Nonce n`, to identify the session. `Prop` corresponds to the second step, the addition of an offer by a server. The offer is represented by an unguessable nonce, `Nonce ofr`. `agl` is the subset of messages representing lists of agents. `isin (Agent C, app (J, del (Agent B, I)))` means that the next server `C` is picked among the agents of `I`, except the first occurrence of `B`, or among a new list of agents `J`. Finally, `reqm` and `prom` are the request message and the proposition message respectively. They are specific to each protocol.

The properties claimed to hold for these two protocols are the following:

- *Data confidentiality*: only the owner of the agent can extract the offers.
- *Non-repudiability*: a shop cannot repudiate an offer once it has been received by the owner of the agent.
- *Forward privacy*: (for P2) none of the identities of the shops can be extracted.
- *Strong forward integrity*: except the last one, offers cannot be modified.
- *Publicly verifiable forward integrity*: (for P1 only) anyone can verify the integrity of a list of offers.
- *Insertion resilience*: no offer can be inserted between two previous offers.
- *Truncation resilience*: the list of collected offers can be truncated only at an offer whose author colludes with the attacker.

Note that forward privacy only means that no one can extract the identity of the shops by looking only at the list of offers. This does not mean that the identity of shops cannot be inferred by other means as described above.

The idea proposed by Asokan, Gülcü and Karjoth for ensuring insertion resilience, truncation resilience and a stronger form of forward integrity than the one proposed by Yee [26] is to add unforgeable dependencies between offers, creating what they call a *chaining relation*:

- The agent starts with a hash of the message made of the identity of the first server to be visited `B`, salted with some random number `k`: `Hash {|Agent B, Nonce k|}`
- Then, within its offer, signed with its private key, each shop must add a hash of the message made of the previous offer `M` together with the next server `C` to be visited: `Hash {|M, Agent C|}`.

The complete definition of `chain B ofr A L C` is as follows for P1:

```
sign B {|Crypt (pubK A) (Nonce ofr), Hash {|head L,Agent C|}|}
```

and as follows for P2:

```
{|Crypt (pubK A) (sign B (Nonce ofr)), Hash {|head L,Agent C|}|}
```

where `head L` denotes the previous offer collected by the agent and `sign` is the signature function defined by `sign B X = {|Agent B, X, Crypt (priK B) (Hash X)|}`. The start of the chaining relation, `anchor`, is defined as a particular case of `chain` by `anchor A n B = chain A n A (cons nil nil) B`.

Hence, if someone wants to modify or insert an offer, for the chaining relation to be preserved, he must be able to modify as well all the following offers, which is made *a priori* impossible by asking the shops to sign their offers with their private keys. The two remaining possible attacks are the deletion of the offers between two offers made by two colluding shops (or the same shop if the agent goes twice to the same malicious shop) or the deletion of all the offers (denial-of-service attack), and the addition of fake offers. So, a shop cannot even modify its own offer unless it colludes with another shop visited later (which may be the same) but, in this case, it must delete all the intermediate offers. For limiting this difficult deletion/truncation problem, Asokan, Gülcü and Karjoth suggest a few solutions like adding several next shops instead of just one.

In our formalization, we do not need to include the salt random numbers r_i used in [2] since the spy is not able to infer the content of encrypted messages if he does not know the inverse of the keys used for encrypting them (`analz` function).

We can know present the formal definitions of the requests and propositions:

```
reqm A r n I B = {|Agent A, Number r, cons (Agent A) (cons (Agent B) I),
  cons (anchor A n B) nil|}
```

```
prom B ofr A r I L J C = {|Agent A,Number r, app (J, del (Agent B, I)),
  cons (chain B ofr A L C) L|}
```

In the request, `A` and `B` (the first server to be visited) are added to the itinerary `I`. In the proposition, `B` is deleted from `I` and a new list of servers `J` is added.

6 Correctness of P1 and P2

For proving the strong forward integrity, the insertion resilience and the truncation resilience, we need to define what is a *valid* chaining relation:

```
inductive "valid A n B" intros
Req: "cons (anchor A n B) nil ∈ valid A n B"
Prop: "L ∈ valid A n B
  ⟹ cons (chain (next_shop(head L)) ofr A L C) L ∈ valid A n B"
```

And to formalize the corresponding attacks, we use the following functions:

- `ith(L,i)` is the `i+1`-th element of `L`.
- `repl(L,i,M)` is the list `L` with its `i+1`-th element replaced by `M`.
- `ins(L,i,M)` is the list `L` with `M` inserted before the `i+1`-th element of `L`.
- `trunc(L,i)` truncates the `i` first elements of `L`.

The three properties are then easily proved by induction on `valid`:

```
lemma strong_forward_integrity: "⟦L ∈ valid A n B; Suc i < len L;
  repl(L,Suc i,M) ∈ valid A n B⟧ ⟹ M = ith(L,Suc i)"
lemma insertion_resilience: "⟦L ∈ valid A n B; Suc i < len L⟧
  ⟹ ins(L,Suc i,M) ∉ valid A n B"
lemma truncation_resilience: "⟦L ∈ valid A n B; Suc i < len L;
  cons M (trunc(L,Suc i)) ∈ valid A n B⟧ ⟹ shop M = shop (ith(L,i))"
```

For the data confidentiality, we first prove that both a request `n` and an offer `ofr` are guarded by the private key of the agent's owner. Then:

```
lemma req_notin_spies: "⟦evs ∈ p1; req A r n I B ∈ set evs; A ∉ bad⟧
  ⟹ Nonce n ∉ analz (spies evs)"
lemma pro_notin_spies: "⟦evs ∈ p1; pro B ofr A r I L J C ∈ set evs;
  A ∉ bad; B ∉ bad⟧ ⟹ Nonce ofr ∉ analz (spies evs)"
```

We also proved that requests and offers are not known by other agents (and not only by the spy as it is commonly done). Although this is not very complicated, this requires to extend the Isabelle libraries.

For the non-repudiability, we proved that the signature scheme is secure:

```
lemma "⟦evs ∈ p1; A ∉ bad; sign A X ∈ parts(spies evs)⟧
  ⟹ ∃ B Y. Says A B Y ∈ set evs & sign A X ∈ parts Y"
```

We would like to point out that, although it was the first time we used Isabelle, thanks to the way we formalized P1 and the power of the Isabelle tactics (although, sometimes, we would like them to be less sensitive to some syntactical aspects), once P1 has been proved, it took us only a few minutes to formalize and prove P2 by essentially changing the definition of `chain`. It is easy to experiment with changes in message formats.

7 Conclusion and future work

Approaches based on proof assistants like Paulson's inductive approach [21] in Isabelle [18] are known to require much effort, typically several days or weeks,

for certifying a protocol. Establishing general protocol-independent theorems like the ones we presented in this paper helps to reduce this development time and also to get a better understanding of protocol problems. To go further, automated theorem provers could be used or tactics developed for automatically proving the conditions of these theorems. In the case of our guardedness condition, it is clear from the examples we give that the proofs follow similar patterns. This is why we expect to define Isabelle tactics for doing that and also for proving the unicity lemmas automatically.

Finally, we did not take into account in our formalizations what some authors call the "oops" rule [23], that is, the fact that, for some protocols like Yahalom, a particular session is not affected by the compromise of other session keys. But, instead of adding the oops rule and doing the proof again, one may look for conditions under which a secrecy property without oops implies the same property with oops. This would provide another important and useful protocol-independent result.

Acknowledgments. This work has been done during my stay at Cambridge (UK) in 2002 thanks to a grant from the INRIA (French National Research Institute in Computer Science and Automatics) and the EPSRC grant GR/R01156/R01 *Verifying Electronic Commerce Protocols*. We also want to thank Larry Paulson and Glynn Winskel for their comments on this paper.

References

1. M. Abadi, M. Burrows, and R. Needham. A logic of authentication. *Proceedings of the Royal Society*, 426(1871):233–271, 1990.
2. N. Asokan, C. Gülcü, and G. Karjoth. Protecting the computation results of free-roaming agents. In *Proceedings of the 2nd International Workshop on Mobile Agents*, Lecture Notes in Computer Science 1477, 1998.
3. G. Bella and L. Paulson. Kerberos version IV: inductive analysis of the secrecy goals. In *Proceedings of the 5th European Symposium on Research in Computer Security*, Lecture Notes in Computer Science 1485, 1998.
4. L. Buttyán, S. Staamann, and U. Wilhelm. Introducing trusted third parties to the mobile agent paradigm. In *Proceedings of the 4th ECOOP Workshop on Mobile Object Systems*, Lecture Notes in Computer Science 1603, 1998.
5. G. Di Caro and M. Dorigo. Mobile agents for adaptive routing. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, 1998.
6. E. Cohen. TAPS: a first-order verifier for cryptographic protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, 2000.
7. V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 2001.
8. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.
9. J. Guttman, J. Herzog, and J. Thayer. Honest ideals on strand spaces. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998.
10. J. Guttman, J. Herzog, and J. Thayer. Strand spaces: Why is a security protocol correct? In *Proceedings of the IEEE Symposium on Security and Privacy*, 1998.

11. J. Guttman, J. Thayer, and L. Zuck. The faithfulness of abstract protocol analysis: message authentication. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001.
12. P. Lee and G. Necula. Research on proof-carrying code for mobile-code security. In *DARPA Workshop on Foundations for Secure Mobile Code*, 1997.
13. G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998.
14. G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.
15. J. Millen and H. Rueß. Protocol-independent secrecy. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2000.
16. J. Millen and H. Rueß. Local secrecy for stated-based models. In *Workshop on Formal Methods in Computer Security*, 2000.
17. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–998, 1978.
18. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/`.
19. D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21:8–10, 1987.
20. L. Paulson. Verifying the SET protocol. In *Proceedings of the 1st International Joint Conference on Automated Reasoning*, 2001.
21. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, 1998.
22. L. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Computer and System Security*, 2(3):332–351, 1999.
23. L. Paulson. Relations between secrets: two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
24. N. Shankar. PVS: Combining specification, proof checking, and model checking. In *Proceedings of the 1st International Conference on Formal Methods in Computer-Aided Design*, Lecture Notes in Computer Science 1166, 1996. `http://pvs.csl.sri.com/`.
25. D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, 2001.
26. B. Yee. A sanctuary for mobile agents. In *Proceedings of the 4th ECOOP Workshop on Mobile Object Systems*, Lecture Notes in Computer Science 1603, 1998.